assignment3

Assignment3: Scheduling Algorithms in xv6

Background

In this assignment, you will modify xv6 to implement three different scheduling algorithms, validate their correctness with test cases, and analyze their behavior.

General Requirements

1. Work with the RISC-V version of xv6.
2. modify the kernel source code to implement the scheduling logic.
3. use test programs to verify the correctness of your implementation.
4. Submit:
   - Modified kernel source code (with comments highlighting key changes).
   - Test programs (`.c` files) and their output.
   - A detailed report explaining your implementation and test results.
5. No cheating is allowed (copy from other student or purely using LLM to generate the codes).

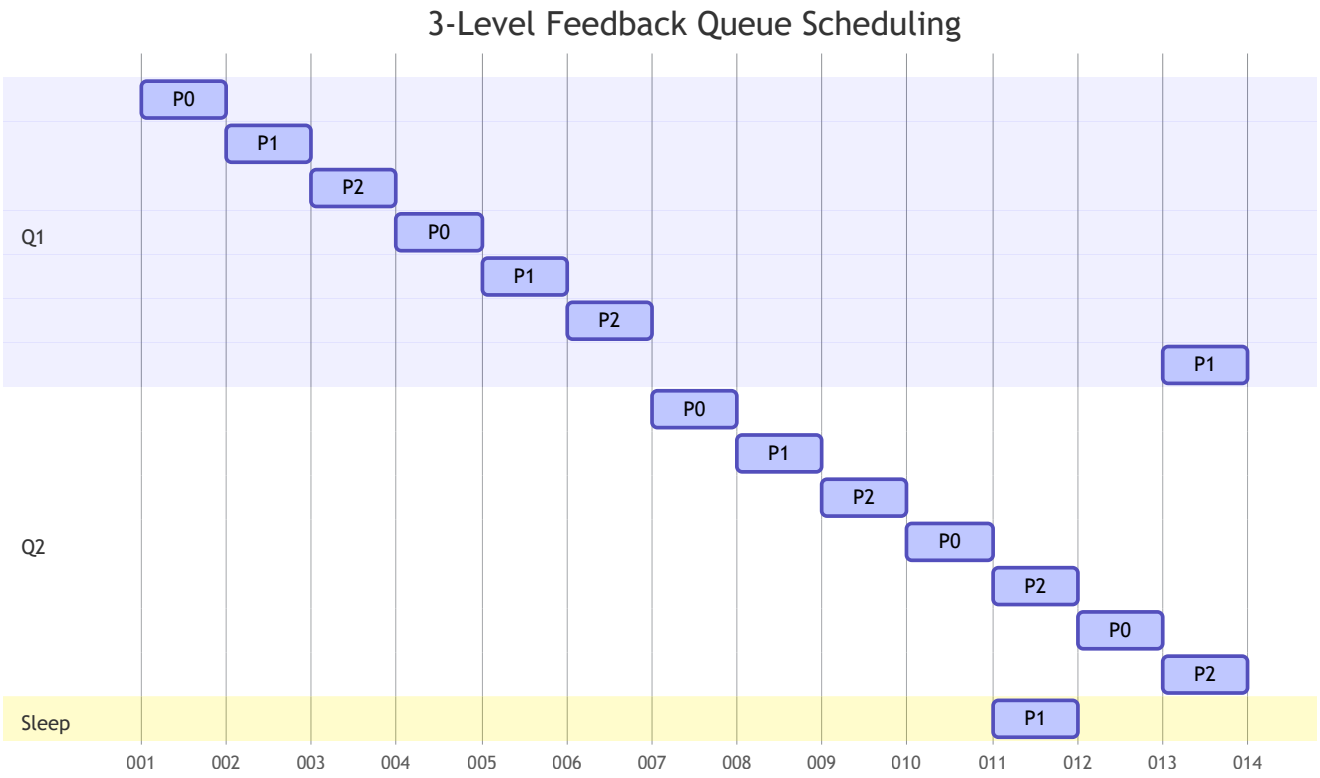
Problem: Implement A Multi-Level Priority Feedback Queue Scheduling

Task Description

Implement a 3-level priority feedback queue scheduler with the following rules:

1. Queues:
   1. Q0 :highest priority with shortest time slice (example : 3 ticks).
   2. Q1 :mid-priority with slightly longer time slice (example : 6 ticks).
   3. Q2 :lowest priority with longest time slice (example : 12 ticks).
2. New processes start in Q0.
3. If a process uses its entire time slice quota, it moves to the next lower queue
   1. (Q0→Q1→Q2).
4. Processes that block on I/O (e.g., `sleep()`) return to their original queue when woken.
5. Every 100 ticks, all processes in Q1 and Q2 are promoted to Q0 to avoid starvation.
6. For processes in same queue, they will be scheduled in Round Robin manner.

Example:



Expected Output

- CPU-heavy process demotes through queues (Q0→Q1→Q2).
- I/O-heavy process stays in high queues after waking:
- **the example below is not the real output, just for illustration.**

```
=== Scheduler Test ===
[CPU-1] starts at t=10 (needs 80 ticks, starts first promote to processing first)
[CPU-2] starts at t=11 (needs 3 ticks)
[CPU-1] running at t=11 (elapsed: 1 ticks)  // Q0 time slice used
[I/O-1] starts at t=11 (I/O-bound)
[I/O-1] finishes I/O operation at t=12, goto sleep
[I/O-1] wakes from I/O 1 at t=14  // Returns to Q0
[CPU-1] running at t=14 (elapsed: 3 ticks)  // Now in Q1
[CPU-2] running at t=15 (elapsed: 1 ticks)  // Q0 time slice used
[CPU-2] finishes at t=18 (elapsed: 3 ticks)
[I/O-1] finishes I/O 2 at t=15
[I/O-1] wakes from I/O 2 at t=17  // Returns to Q0
[CPU-1] running at t=17 (elapsed: 7 ticks)  // Now in Q2
[I/O-1] finishes I/O 3 at t=18
[I/O-1] wakes from I/O 3 at t=20
[I/O-1] finishes at t=21
[CPU-1] finishes at t=102
=== Test Complete ===
```

Failure Condition:

- If CPU-1 does not demote or I/O-1 is demoted, MLFQ logic is incorrect.

Bonus problem:

There will be two bonus problems, each worth 10 extra marks.

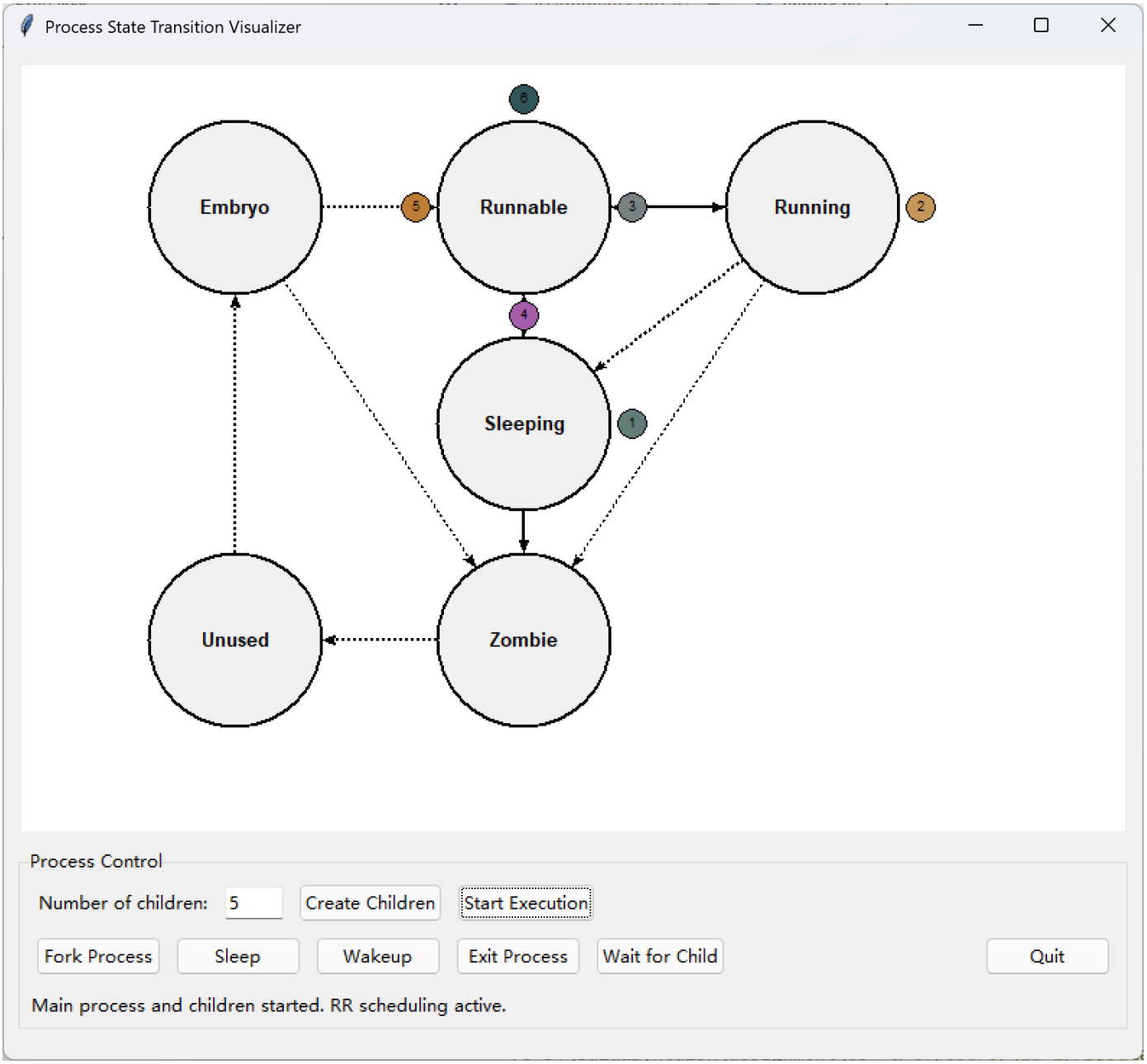
1. **Bonus problem 1**:

- Implement a real time visualizer to show the scheduling process and the life cycle of each process.
    1. demonstrate how the process state transfer between typical 5 stages in real time
    2. All actions in the visualizer should be real-time, reflecting the immediate changes in the scheduler in XV6 operating system.

2. **Bonus problem 2**:

- Base on bonus problem 1, add interactive features to the visualizer.
    1. allow users to pause and resume the scheduling process.
        1. When pause, the scheduling should keep the current process keep running until it finishes or blocks on I/O.
        2. When resume, the scheduling should continue from the point where it was paused.
    2. allow users to add/exit sub-processes dynamically.
    3. display the current time and the state of each process in the visualizer.
    4. For this bonus problem, you can modify any thing you need in xv6-riscv to support the interactive features of the visualizer.

Example for Bonus problem:



- graphical state transition diagram and life cycle of each process
- interactive control features (pause, resume, add/exit sub-processes)

Reports

Use the provided LaTeX template to submit your report. In the report, include:

- A detailed description of your scheduling algorithms .
- Explanations of key code modifications (e.g., data structures, algorithms).
- Test cases and their expected outputs.
    - You can use the provided test programs to verify the correctness of your implementation and modifications to provided code is allowed.
        - You can add more test cases to demonstrate and verify the correctness of your implementation.

- Challenges faced during implementation and how you overcame them.
- A comparison of your scheduler's performance with the original RR scheduler (e.g., throughput, response time).

Grading:

Basic 100 mark will grading under the following criteria:

- **Correctness**:
  - Scheduling logic adheres to the rules, mechanism of the scheduling algorithm and its behavior in different scenarios satisfy the requirements.
  - Life cycle of a process in the scheduling algorithm fits the requirements.
- **Test Cases**:
  - Tests results fit expectation.
- **Documentation and Presentation**:
  - Good documentation with:
    - Clear language to describe the required topics.
    - Diagrams/animations/PlantUML/Python demos or other tools to demonstrate the process life cycle.

Submission Requirements

Submit all materials by the deadline (11/12/2025). Good luck!

Additional Tips

Scheduling Algorithms in xv6

XV6 is a simple Unix-like operating system, uses a round-robin scheduling algorithm by default, where all processes share the CPU equally. Some info on how the scheduling of XV6 is implemented can be found in Chapter 7 of the ref-book.

Key Modifications Hints

- Add `queue_level` (0–2), `remaining_ticks` (time slice quota left), and `last_promote` to `struct proc`.
- Modify the scheduler to check queues in order Q0→Q1→Q2.
- In the timer interrupt, decrement `remaining_ticks`; if 0, demote the process and trigger `yield()`.
- Add a periodic check (every 10 ticks) to promote all Q1/Q2 processes back to Q0.

How to get the xv6-riscv source code

1. Clone the xv6-riscv repository:
   `git clone https://github.com/mit-pdos/xv6-riscv.git`

How to update qemu to 7.2.0

The standard xv6-riscv requires qemu 7.2.0 or higher. Here's how to update qemu:

```
wget https://download.qemu.org/qemu-7.2.0.tar.xz
tar xvf qemu-7.2.0.tar.xz
cd qemu-7.2.0
./configure --target-list=riscv64-softmmu --prefix=/usr/local
make -j$(nproc)  # Compile using all available CPU cores
sudo make install
```

How to run xv6-riscv environment

1. Run xv6-riscv with: `make qemu`
    1. This will compile the xv6-riscv kernel and run it in qemu.
        1. For more qemu execution options, you can check the Makefile.
    2. if everything goes well, you should see the info like this in the qemu window:

    ```
    csc3150@csc3150:~/hw3/xv6-riscv$ make qemu
    qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0
    xv6 kernel is booting
    hart 2 starting
    hart 1 starting
    init: starting sh
    $
    ```

2. The xv6-riscv will run in the qemu window.
3. Run the test program in xv6-riscv:

    ```
    $Q3_test
    ```

4. Exit qemu with `Ctrl+A` followed by `X`.

How to install and enable GUI in current vm:

Because our vm is running in getty only mode, you need to install and enable GUI to see the visualizer.

```
sudo apt install xfce4 xfce4-goodies -y
sudo systemctl set-default graphical.target
sudo systemctl enable --now lightdm
```

This will install xfce4 and its dependencies, and enable lightdm.